Costruzione di una schedule ciclica

- Tre decisioni fondamentali:
  - scelta della *dimensione del frame*
  - partizionamento dei job in sotto-job (slicing)
  - allocazione delle slice nei frame

- In generale, non sono decisioni indipendenti:
  - lo slicing semplifica l'allocazione ma aumenta l'overhead

Partizionamento dei job – L'esempio rivisitato

- Dopo lo slicing di T3:

  T1 =(4, 1, 4) T31=(20, 1, 20) T33=(20, 1, 20)

  T2 =(5, 2, 5) T32=(20, 3, 20)

- Vincoli:

  (1) $m \leq D_i \ \forall i$ → $m \leq 4$

  (2) $m \geq c_i \ \forall i$ → $m \geq 3$

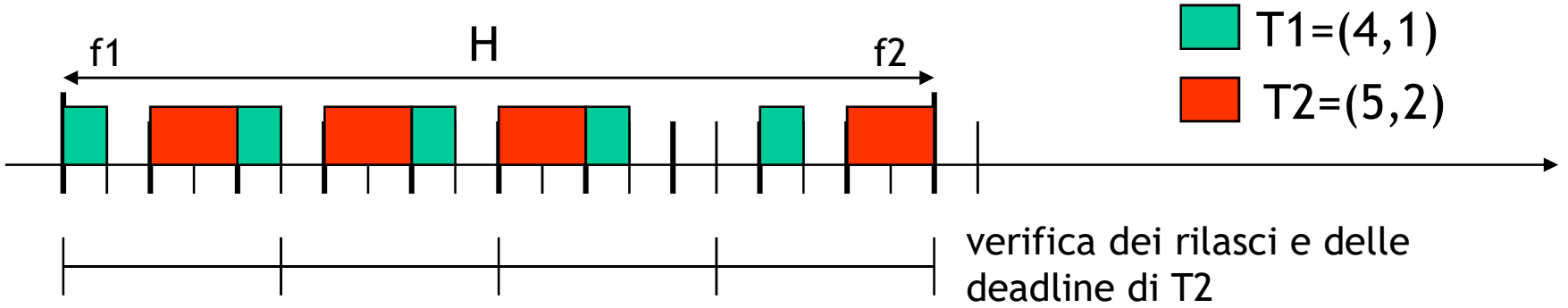
  (3) m divide H → m=2,4,5,10,20

  (4) $2m - MCD(m,p_i) \leq D_i \ \forall i$ → valutiamo solo per m=4 !

- Con m=4: 8-MCD(4,4) ≤ 4; 8-MCD(4,5) ≤ 5; 8-MCD(4,20) ≤ 20
- Il secondo vincolo diventa: 8-1 ≤ 5 !! → m=4 non è una dimensione di frame corretta
- Il vincolo H/m impone di studiare uno slicing per m=2

Partizionamento dei job – Esempio (segue)

- Dimensione frame: m=2 (ovviamente soddisfa vincoli)
- Occorre indagare un diverso *slicing* di T3
- Allocazione parziale di T1 e T2 sul ciclo maggiore:



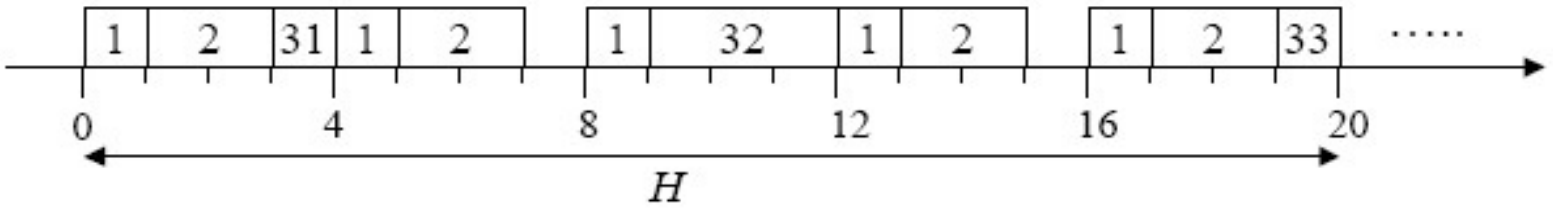
- Dopo questa allocazione parziale si vede come solo alcuni partizionamenti di T3 siano possibili
  - Ad es. (1,1,2,1), ma non (1,2,1,1)

Partizionamento dei job – Esercizio modificato

- Esercizio: verificare la seguente schedule per l'insieme di task periodici:

T1 =(4, 1, 4)
T2 =(5, 2, 7)
T31=(20, 1, 20)
T32=(20, 3, 20)
T33=(20, 1, 20)

Vincolo (4): $2m - MCD(m,p_i) \leq D_i \ \forall i$



Osservazioni

- Calcolo della dimensione del frame e partizionamento dei job sono operazioni complesse e non indipendenti
  - Negli esempi abbiamo a che fare con 3 task periodici. Con 20 task?
  - Per minimizzare l'overhead: dimensione di frame massima e minor numero di partizionamenti (forme di *preemption pianificate*)
  - Complessità spostata offline
- Come otteniamo la schedule vera e propria in modo automatico (algoritmo di sintesi)?
- Anche «limitate» variazioni nei parametri dei task possono determinare una modifica radicale della schedule ricavata

Executive ciclico

- Modifica del codice dello scheduler per far sì che le decisioni di scheduling siano prese *solo all'inizio dei frame*

- Ipotizziamo l'esistenza di un timer precaricato che genera un interrupt con periodo f (dimens. frame)

- Input per l'executive:
  - Schedule memorizzata: L(k) per k=0,1,...,F-1
  - Coda dei job aperiodici
- L(k)= blocco di scheduling

Executive ciclico

```
Task CyclicExecutive:
t:=0 /* tempo attuale */; k=0 /* frame attuale */;
CurBlock:=empty;
do forever:
   sleep fino al prossimo clock interrupt; /* istante k·f */
   if <job in CurBlock non completato> invoca FmOverrunHandler;
   CurBlock:=L(k);
   k:=k+1 mod F;  t:=t+1;
   if <job in CurBlock non rilasciato> invoca RelErrorHandler;
   risveglia il server dei task periodici per i job in CurBlock;
   sleep fino a che il server dei task periodici completa;
   while <coda dei job aperiodici non vuota>
     esegui il primo job in coda;
     rimuovi il job appena completato;
   end while;
end do;
end CyclicExecutive;
```

Osservazioni

- Importante: il timer *non* viene ricaricato intervallo per intervallo; --> misura il tempo in modo regolare senza derive
- Verifiche dei vincoli temporali su rilasci e deadline integrate negli istanti di decisione
- Ogni frame può eseguire uno scheduling block
  - tipicamente l'executive assegna a un thread (periodic server) o a thread specifici l'esecuzione dei job/slice dello scheduling block

- Per utilizzare l'idle time in modo controllato si può prevedere un aperiodic_task_server(), che esegua a priorità inferiore rispetto all'executive e al periodic_task_server()

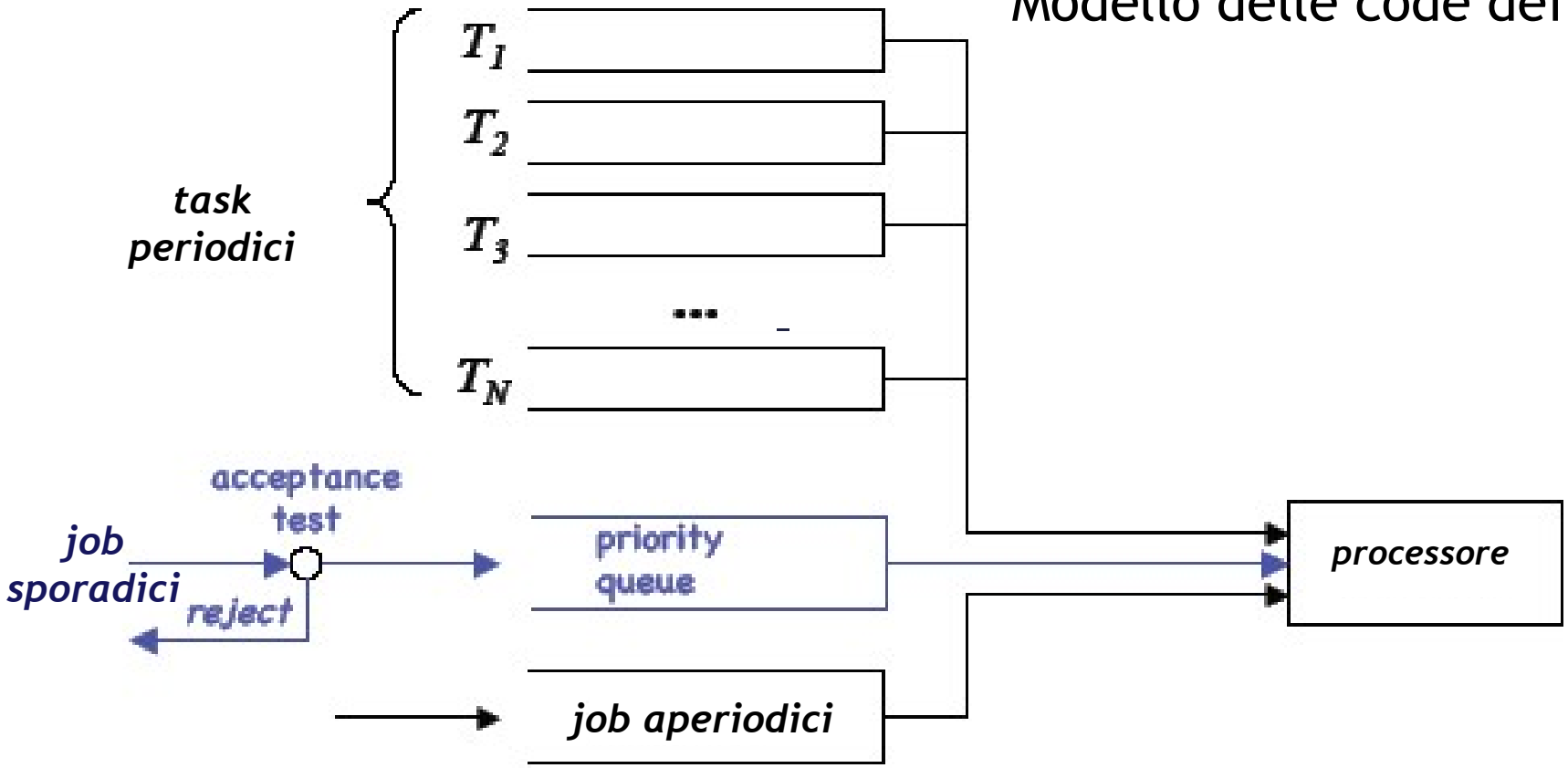
Integrazione di carico aperiodico e sporadico in scheduling clock-driven

- Ipotesi:
- Il carico associato ai task periodici hard real-time è prevalente
  - altrimenti l'approccio clock-driven non è quello appropriato!

- Estensione:
  - integrazione di task aperiodici -> soft real-time
  - Integrazione di task sporadici -> hard real-time; richiede *accettazione*

Scheduling clock-driven di task periodici con integrazione di job sporadici e aperiodici

Modello delle code dei task



Job aperiodici

- Spesso schedulati in background, non presentano deadline hard
- I job aperiodici sono attività di risposta ad *eventi esterni*
- Obiettivo progettuale: riduzione del *tempo di risposta* medio
- Approccio: *Slack stealing* (Lehoczky, 1987)

Slack stealing

- *Slack stealing*: esecuzione dei job aperiodici prima dei job periodici quando possibile
- Ipotesi: ogni job o slice periodico è schedulato in un frame che termina entro la sua deadline
  - x_k: tempo totale allocato ai job nel frame k
  - $f-x_k$: slack time all'inizio del frame k

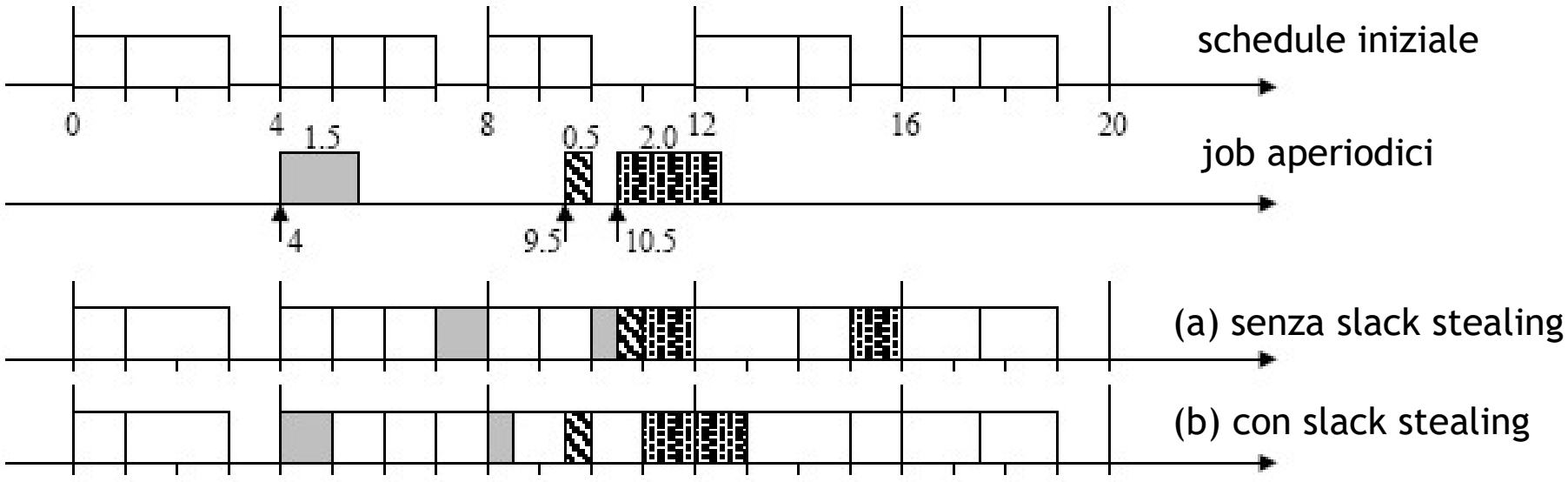
- Dopo che y unità di slack time sono state già utilizzate, è possibile eseguire job aperiodici nel frame fino a quando vale: $f-x_k-y>0$

Slack stealing

- Se i job aperiodici si esauriscono con $f-x_k-y>0$, l'executive pone in esecuzione il *periodic server* per il blocco di scheduling L(k)
- Al termine di ogni job in L(k), se $f-x_k-y>0$ l'executive verifica l'eventuale presenza di nuovi job aperiodici in coda
- Lo slack è rimasto invariato (o è aumentato se $C_i(k)<WCET$)
- Job aperiodici non completati all'interno del frame vengono nuovamente inseriti nella coda dei job aperiodici → preemption

Slack stealing

- Esempio:



Tempi di risposta medi:

$T_a=(6.5+1.5+5.5)/3=4.5$ $T_b=(4.5+0.5+2.5)/3=2.5$

Slack stealing

- Realizzazione:
  - lo slack time iniziale in ciascun frame può essere precalcolato e memorizzato in tabella con L(k)
  - l'executive deve tenere traccia del tempo dedicato ai job aperiodici ed aggiornare lo slack time disponibile
  - occorre un timer da settare allo slack time disponibile all'inizio del frame
- Problemi:
  - timer con adeguata risoluzione;
  - se lo slack time è scarso, il timer è sufficientemente accurato? vale la pena mettere in esecuzione il job?
  - in pratica applicabile con slack e parametri ≥10 ms

Job sporadici

- Job con caratteristiche asincrone e istanti di rilascio non prevedibili ma che *richiedono garanzie real-time*
- Per poter fornire garanzie, è necessario formulare alcune *ipotesi* sulle caratteristiche dei job:
  - tempo minimo di interarrivo noto,
  - tempo di esecuzione WCET noto o dichiarato all'arrivo insieme alla deadline,
  - eventuale conoscenza a priori del numero e delle caratteristiche di tutti i possibili job sporadici
- In assenza di queste conoscenze a priori, è necessario prevedere un modulo di accettazione: job sporadici potranno essere rifiutati; quelli ammessi nel sistema saranno garantiti

Job sporadici in sistemi clock-driven

- Situazione frequente:
  - uno o pochi job sporadici con caratteristiche note a priori Js(D,e)
  - i job possono essere implicitamente garantiti dalla presenza di sufficiente slack time, noto, nell'iperperiodo H
- Se i job sporadici non hanno caratteristiche note o sono numerosi:

  (a) approccio sbagliato?

  (b) è necessario prevedere accettazione
  - il job non garantito viene rifiutato;
  - può essere risottoposto con requisiti inferiori di QoS (es. D maggiore) oppure come job aperiodico eseguibile in modo «best effort»

Job sporadici

- Problema: gestione dei job sporadici nel caso più generale

- Idea base per la gestione dei job sporadici nell'ambito di uno scheduling clock-driven:
  - Usiamo la *trama sincrona* dei frame per garantire le deadline dei job sporadici

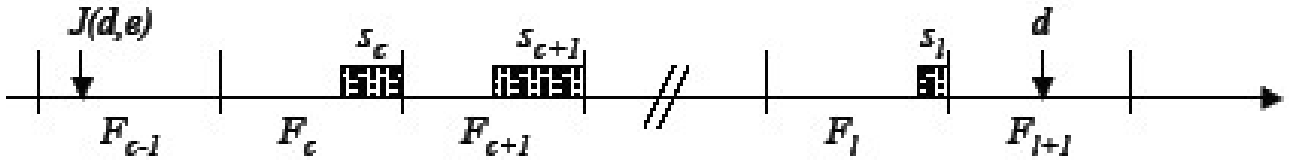
Job sporadici

- Caratteristiche: *deadline hard*, istanti di rilascio e tempi di esecuzione non noti a priori, tempo massimo di esecuzione noto all'istante di arrivo, revocabili

- Richiedono *test di accettazione*: il job sporadico viene accettato se è schedulabile e non compromette la schedulabilità dei job periodici o di altri job sporadici già accettati
- *Js(d,e)* è *schedulabile se* $\mathbf{S_c(t,l)} \geq \mathbf{e}$, ove $S_c(t,l)$ è lo slack corrente totale dall'istante attuale *t* fino al frame *l* che precede la deadline d (*l*+1 termina dopo *d*)
- La *garanzia* di Js (hard), richiede che ne venga verificato il completamento al più entro la deadline

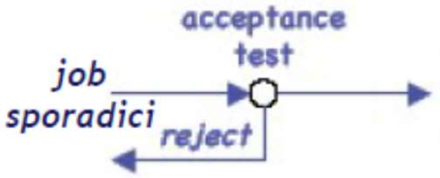
Job sporadici

- test di accettazione:



if $S_c(t,l) < e$ **then** reject job;
else
    accept job;
    schedula esecuzione;
end;

- Più job sporadici <u>in attesa di accettazione</u> possono essere valutati *in ordine EDF*



Scheduling dei job sporadici accettati

- Scheduling statico:
  - Scheduling nel frame corrente di una slice (più grande possibile) del job accettato
  - Scheduling della parte rimanente il più tardi possibile, eventualmente in più slice
- Meccanismo:
  - Le slice del job accettato sono appese, nei frame in cui vengono schedulate, alla lista dei job periodici
- Problema: arrivo di successivi job sporadici
- Alternative:
  - Rescheduling all'arrivo di nuovi job
  - Scheduling priority-driven dei job sporadici (ad es. EDF)

Scheduling EDF dei job sporadici accettati

- Soluzione realistica: valutazione e accettazione dei job sporadici all'inizio del frame, in modo EDF
- La coda dei job sporadici accettati viene mantenuta ordinata in modo EDF
- I job sporadici possono essere eseguiti, ad esempio, al termine dei job periodici del frame (non c'è vantaggio ad anticipare)
- In presenza di job sporadici e aperiodici devono essere favoriti quelli sporadici, tuttavia è possibile aggiungere lo slack stealing per gli aperiodici
- Lo scheduling dei job sporadici accettati in modo EDF è *ottimo tra gli algoritmi che eseguono il test di accettazione all'inizio del frame*

Realizzazione del test di accettazione

- Ipotesi: test di accettazione *all'inizio del frame (➔ EDF ciclico)*
- Test per il job Js(d,e):
  (1) Determinare se lo slack time nei frame che precedono *d* è sufficiente: $S_c(t,l) \geq e$
  (2) Verificare che altri job sporadici già accettati con deadline successiva a *d* non completino in ritardo
- Se una delle due condizioni è falsa → reject Js(d,e)
- E' utile disporre dello slack corrente totale tra ogni coppia di frame *i* e *k*, precalcolandolo per i job periodici (richiede una matrice triangolare, $O(|F|^2)$)
- Se *i* e *k* appartengono a due cicli maggiori diversi *j* e *j'*:
  $S(i+(j-1)F, k+(j'-1)F)=S(i,F)+S(1,k)+(j'-j-1)S(1,F)$

Realizzazione del test di accettazione

- Allo slack time precalcolato in base ai job periodici occorre sottrarre la quota ancora da eseguire dei job che avranno priorità sul job corrente nell'ordinamento EDF
- Se Js(d,e) è accettato, i job J_k con deadline $d_k>d$ saranno ritardati; deve valere: $s_k=s_k-e\geq 0 \quad \forall k$ tale che $d_k>d$
- Prima di accettare Js occorre quindi verificare che questi job completino ancora entro le proprie deadline
- Al crescere del numero di job sporadici, il sistema diventa complesso e perde i vantaggi di semplicità e affidabilità dell'approccio clock-driven
- → in pratica si applica per pochi job che non rientrano nel paradigma periodico

Problemi realizzativi dell'approccio clock-driven: Frame overrun

- Possibile? I task non erano stati accuratamente garantiti con sofisticate analisi offline?
- Cause: WCET imprecisi, guasti transienti
- Possibili politiche per overrun sporadici:
  - abortire il job all'inizio del nuovo frame
  - preemption (se non in sez. critica); la parte residua è eseguita come job aperiodico nei frame successivi
  - continuare l'esecuzione; i job del frame successivo sono ritardati; ad es. consentire al più lo slack time del frame successivo
  - mix; comunque tracciare evento
  - con overload prolungato, garantire *(k* su *m)* job per uno o più task?
- La politica più adatta dipende dalla funzione valore dei task per l'applicazione specifica

Problemi realizzativi dell'approccio clock-driven: Cambiamenti di modo

- Per riconfigurazione del sistema
- Normalmente è necessario un lasso di tempo per caricare nuove tabelle e codice dei job e per allocare memoria
- → *mode change job* ! Può essere hard o soft
  - → aperiodic mode change
  - → sporadic mode change
- Quando intervenire (sui singoli job o su insiemi di job)?
  - al termine del ciclo maggiore
  - al termine del frame
  - immediatamente (abort del job in esecuzione e dei job partizionati)

Problemi realizzativi dell'approccio clock-driven: Cambiamenti di modo

- Job di task appartenenti ad entrambi i modi restano in esecuzione
- E' necessario conoscere la rilevanza dei job aperiodici e sporadici nel nuovo *modo*
- Ad es. (ipotesi): i job aperiodici possono essere abortiti, mentre gli sporadici vanno completati

- In generale esiste un problema di *transitorio*; obiettivo: minimizzazione della durata del transitorio mantenendo la consistenza del sistema

Pregi e difetti dello scheduling clock-driven

- Vantaggi:
  - + semplicità concettuale
  - + assenza di anomalie
  - + predicibilità di funzionamento
  - + minimo overhead
  - + assenza o limitazione di preemption non pianificate

Pregi e difetti dello scheduling clock-driven

- Limitazioni:
  - rigidità
  - difficoltà di modifica e manutenzione
  - richiede conoscenza a priori (istanti di rilascio dei task periodici, combinazioni di task periodici nei diversi modi, etc.)
  - integrazione problematica di componenti di carico dinamiche